APPENDIX A

Mathematical Considerations Concerning Converting Numbers from Binary Coded Decimal to Binary

The purpose of this Appendix A is to explain that respondents' contribution is merely a mathematical expression or algorithm and as such can be expressed in a number of vastly different mathematical forms and terminologies. Further by utilizing simple examples, respondents' contribution will be recognized by those familiar with mathematics to be a trivial modification of standard algorithms for converting numbers from Binary Coded Decimal to Binary notation.

Summary

The First Example illustrates a conventional, straightforward multiplication method for converting a decimal number into a corresponding binary number.

The Second Example illustrates the convenience which arises in solving the same problem set forth in the First Example when the well-known mathematical principle of factoring is employed before the initial multiplication steps are carried out.

The Third Example illustrates a detailed mathematical analysis of the generalized algorithm for converting numbers from Binary Coded Decimal to Binary form. This analysis and methodology clearly brings out the inherent shifting property of numbers which is useful in effecting multiplication in computing apparatus and in converting numbers from Binary Coded Decimal to Binary notation.

The Fourth Example illustrates the fact that the algorithm embraced by respondents' patent application claims 8 and 13 may be expressed either in the Truth Table form of propositional calculus[1] or by Boolean algebra.[2] These latter forms of expression are everyday tools of a mathematical logician or logic designer of computer hardware.

FIRST EXAMPLE: Brute Force Multiplication

The decimal number 953, can be represented as follows, recognizing that the 9 is in the one-hundreds position and the 5 is in

---

[1] P. Rosenbloom, The Elements of Mathematical Logic 28 (1950).

[2] See G. Boole, The Laws of Thought (1854).

the tens position thus giving them weights of one hundred and ten, respectively:

Expression 1 $(9 \times 100) + (5 \times 10) + 3 = 953$

The left-hand portion of Expression 1 can be represented in the binary number system as follows:

Expression 2 $(1001 \times 1100100) + (0101 \times 1010) + (0011) = 953$

A straightforward technique of multiplication of binary and decimal numbers respectively can be carried out in the commonly known manner as follows:

Expression 3	Binary	Decimal Equivalent
	1100100	100
	X 1001	X 9
	1100100	
	1100100	
3(a)	1110000100	900
	1010	10
	X 0101	X 5
	1010	
	1010	
3(b)	110010	50

This results in two partially converted portions of information, 3(a) and 3(b). When performed on a computer both 3(a) and 3(b) must be stored in the computer memory and then retrieved after all operations have been completed so that the partial results can then be added together to give the final result as follows:

Expression 4	Binary	Decimal Equivalent
	1110000100	900
	110010	50
	+ 0011	+ 3
	1110111001	953

SECOND EXAMPLE: Classical Factoring

Instead of performing the calculations in accordance with the steps of the First Example, if Expression 1 is first factored by removing a ten from the tens and hundreds position, then this same problem may be expressed and solved as follows:

Expression 5 $(9 \times 10 + 5) \times 10 + 3 = 953$

As before, the decimal numbers of the left-hand side of Expression 5 may be converted into their binary equivalents as follows:

Expression 6 $(1001 \times 1010 + 0101) \times 1010 + 0011 = 953$

According to the common mathematical practice, one begins solving the problem in Expression 6 by first working upon the numbers appearing within the parentheses as follows:

Expression 7

Binary	Decimal Equivalent
1010	10
× 1001	× 9
1010	90
1010	
+ 0101	+ 5
1011111	95

Next, the above results within the parentheses are operated upon by multiplying the sum (95) by the factor (10) and adding the remaining units digit (3) as follows:

Expression 8

Binary	Decimal Equivalent
1010	10
× 1011111	× 95
1010	50
1010	90
1010	
1010	
1010	
1010	
+ 0011	+ 3
1110111001	953

As can be seen, by factoring out ten from the numbers within the parentheses of Expressions 5 and 6, each operation directly utilizes the results from the previous operation. The work may now be carried out without the necessity of writing down the partial results 3(a) and 3(b), or if performed by a computer instead of by hand, without storing these partial results in the computer storage facility. Thus respondents' mathematical contribution is merely a routine application of the well-known mathematical technique of factoring[3] so as to create a slightly different algorithm. The court below was wrongly impressed with the "discovery" that by adding a binary "1" in two specific places, conversion can be accomplished.[4] The fact that the binary "1" is added to the two specific places is a direct result of factoring the binary number 1010 (*i.e.*, decimal 10) out of Expression 5. When, once again, the binary number 1010 is multiplied back into the factored expression a binary 1 is added to two specific places. This is graphically illustrated by Expression 8 where the binary number 1010 is added six times and progressively shifted over to the left. This then results in the addition of a binary "1" to two specific bit positions of the binary number 3 in Expression 8.

This example illustrates the complete independence of respondents' mathematical contribution from the hardware components that may be included in any given general-purpose digital computer. It is misleading to recite language in patent claims directed to computer hardware components or operations when describing a simple algorithm or mathematical expression.

---

[3] "It is more convenient to adopt the less direct method now to be described. This is best illustrated by an example; thus, suppose the number 0.431 is to be converted to the binary scale. This can be written:

$$10^{-3}((4.10 + 3)\ 10 + 1)$$

The process can be analyzed as follows:

Multiply the first digit of the number by 10 (1010 in binary scale), add in the next digit and multiply by 10, add in the next digit, *etc.* This process is repeated until all the digits have been used." A. Booth & K. Booth, Automatic Digital Calculators 190–91 (2d ed. 1956).

[4] "It will aid in understanding the terms of claim 13 to state that the discovery which is the essence of appellants' invention was that, for each bit position in each BCD digit, conversion can be accomplished by adding a binary '1' to two specific positions of the next lesser significant BCD digit and that in normal notation 'significance' of bits or digits increases to the left, e.g., in the decimal number '53' the 5 is more significant than the 3. Thus conversion can be carried out by the steps of (1) testing each BCD

THIRD EXAMPLE:
Analysis of Generalized Mathematical Algorithm for Binary Coded Decimal to Binary Conversion

Number Representations: Positionally coded numbers to any base, for example Decimal, Binary Coded Decimal and Binary, may be graphically represented as follows, with the more significant digit positions increasing from right to left:

Decimal

D_n	...	D_i	...	D_1	D_0

D_i *are Decimal Digits*

Binary Coded Decimal (BCD)

nd_3	nd_2	nd_1	nd_0	...	id_3	id_2	id_1	id_0	...	od_3	od_2	od_1	od_0

id_i *are Binary Digits*

Binary

b_m	...	b_i	...	b_2	b_1	b_0

b_i *are Binary Digits*

A number N may then be expressed as follows:

Decimal: $$N_{10} = \sum_{i=0}^{n} D_i \times 10^i$$

Binary Coded Decimal: $$N_{10} = \sum_{i=0}^{n} \sum_{j=0}^{3} id_i \times 2^j \times 10^i$$

Binary: $$N_2 = \sum_{i=0}^{m} b_i \times 2^i$$

The mathematical basis of and method for conversion between any different number bases, for example decimal to binary conversion, may be derived by analyzing the expanded generalized expressions.

---

bit position for all but the least significant BCD digit and (2) for each '1' bit detected, adding a binary '1' to each of two specified bit positions in the next lesser significant BCD digit." Application of Benson & Tabbot, 441 F.2d 682, 688 (C.C.P.A. 1971).

For corresponding decimal and binary numbers (N_{10} and N_2) the expanded expressions are:

$$N_{10} = D_0 \times 10^0 + D_1 \times 10^1 + \ldots + D_n \times 10^n = N_2$$

$$N_2 = b_0 \times 2^0 + b_1 \times 2^1 + \ldots + b_m \times 2^m$$

Hence $b_i = f_i(D_0, D_1, \ldots, D_n)$ for converting decimal to binary.

Expanding the decimal expression we have:

$$N_{10} = D_0 \times 10^0 + D_1 \times 10^1 + \ldots + D_n \times 10^n$$

Factoring a 10 at each level of the expansion yields:

$$N_{10} = D_0 + 10(D_1 + 10(D_2 + \ldots + 10(D_{i-1} + 10(D_i + 10(D_{i+1} + \\ + \ldots + 10(D_{n-1} + 10(D_n)) \ldots))$$

Analyzing the relationship between the i^{th} element and the $i-1^{st}$ element of this last expression we have:

$$10D_i + D_{i-1}$$

That is, 10 times the more significant digit (i^{th}) is added to the lesser significant digit ($i-1^{st}$). Thus:

$$10(id_3 \times 2^3 + id_2 \times 2^2 + id_1 \times 2^1 + id_0 \times 2^0) + \\ + ((i-1)d_3 \times 2^3 + (i-1)d_2 \times 2^2 + (i-1)d_1 \times 2^1 + (i-1)d_0 \times 2^0)$$

For a digital computing machine in which numbers are expressed in binary notation, the following statements are true and need no proof: (*Note:* Subscripts represent the number base.)

(1) The decimal number ten $(10)_{10}$ is represented as $(1010)_2$

and

(2) Multiplication of a number by 2^n is equivalent to shifting the number to the left by n binary digit positions.

Therefore, the expression

$$[10 \times (id_3 \times 2^3 + id_2 \times 2^2 + id_1 \times 2^1 + id_0 \times 2^0)] + D_{i-1}$$

becomes

$$[id_3 \times 2^3 \times 10 + id_2 \times 2^2 \times 10 + id_1 \times 2^1 \times 10 + id_0 \times 2^0 \times 10] + D_{i-1}$$

or

$id_3 \times (1010_2$ Shifted left 3 positions)
$+ id_2 \times (1010_2$ Shifted left 2 positions)
$+ id_1 \times (1010_2$ Shifted left 1 position)
$+ id_0 \times (1010_2$ Shifted left 0 positions)
$+ D_{i-1}$

To clearly illustrate the shifting property of numbers, let us assume the summation has reached and included the D_{i-1} term. The general procedure to be continued is then to test each id_j (j = 0, 1, 2, 3) and

Step 1
if id_0 is a 1, then add 1010 to the sum; and

Step 2
if id_1 is a 1, then add 10100 to the sum; (or 1010 to a sum that has been shifted right one position); and

Step 3
if id_2 is a 1, then add 101000 to the sum; (or 1010 to a sum that has been shifted right two positions); and

Step 4
if id_3 is a 1, then add 1010000 to the sum; (or 1010 to a sum that has been shifted right three positions).

In each case, if id_j is a 0, then no addition to the summation is necessary, i.e. add 0000 . . .

Thus, the example given by respondents of converting $(53)_{10}$ from its Binary Coded Decimal form to Binary may be seen to merely involve the shifting property of numbers and not a "discovery" that conversion from BCD to Binary may be accomplished by adding a binary "1" to two specific places.

$$(53)_{10} = (0101\ \ 0011)_{BCD}$$

"5" "3"

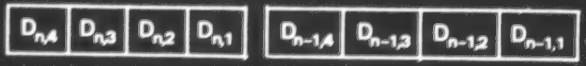
0101		0011	
	+	1010	*Step 1*
	+	00000	*Step 2*
	+	101000	*Step 3*
	+	0000000	*Step 4*
		0110101	

Hence $(0101\ \ 0011)_{BCD} = (0110101)_2$

FOURTH EXAMPLE:
A Typical Logic Designer's Expression of Respondents' Algorithm

The algorithm of Claims 8 and 13 can be represented in the Truth Table form of propositional calculus as follows:

1. Designate as $D_{n,i}$ the i^{th} bit of the n^{th} BCD grouping of bits in the BCD expression:



2. Form a Truth Table which shows the value of the modified (binary-converted) bits as a function of the possible combinations of $D_{n,1}$ and $D_{n,2}$ and $D_{n,3}$ and $D_{n,4}$. Now, by inspection of the Truth Table, yet another representation of respondents' algorithm can be given by the Boolean formulae for each of the modified bits D_i which may be written as follows:

$$D_{n-1,1}\ (modified) = D_{n-1,1}$$

$$D_{n-1,2}\ (modified) = D_{n-1,2} \cdot (\overline{D_{n,4}} \cdot \overline{D_{n,1}} + \overline{D_{n,3}} \cdot \overline{D_{n,2}} \cdot \overline{D_{n,1}}) + \overline{D_{n-1,2}} \cdot (\overline{D_{n,4}} \cdot D_{n,1} + \overline{D_{n,3}} \cdot \overline{D_{n,2}} \cdot D_{n,1})$$

The Boolean formulae for each of the modified bits $D_{n-1,2}$ *(modified)* when written out in this fashion are the form of the algorithm used by the logic designer of hardware to implement this algorithm in computer hardware.

The Truth Table illustrates respondents' algorithm in the classical form of mathematical logic. In the manner demonstrated above, the Boolean formulae for each of the modified bits can likewise be written out, by inspection, from the Truth Table. The full details of this example have been omitted for the sake of simplicity, since such details would be obvious to one skilled in the art of logic design.

FOLD*OUTS ARE TOO LARGE TO BE FILMED

APPENDIX B

The following are the two claims remaining in respondents' application:

8. The method of converting signals from binary coded decimal form into binary which comprises the steps of

(1) storing the binary coded decimal signals in a reentrant shift register,

(2) shifting the signals to the right by at least three places, until there is a binary "1" in the second position of said register,

(3) masking out said binary "1" in said second position of said register,

(4) adding a binary "1" to the first position of said register,

(5) shifting the signals to the left by two positions,

(6) adding a "1" to said first position, and

(7) shifting the signals to the right by at least three positions in preparation for a succeeding binary "1" in the second position of said register.

13. A data processing method for converting binary coded decimal number representations into binary number representations comprising the steps of

(1) testing each binary digit position *i*, beginning with the least significant binary digit position, of the most significant decimal digital representation for a binary "0" or a binary "1";

(2) if a binary "0" is detected, repeating step (1) for the next least significant binary digit position of said most significant decimal digit representation;

(3) if a binary "1" is detected, adding a binary "1" at the (i + 1)th and (i + 3)th least significant binary digit positions of the next lesser significant decimal digit representation, and repeating step (1) for the next least significant binary digit position of said most significant decimal digit representation;

(4) upon exhausting the binary digit positions of said most significant decimal digit representation, repeating steps (1) through (3) for the next lesser significant decimal digit

representation as modified by the previous execution of steps (1) through (3); and

(5) repeating steps (1) through (4) until the second least significant decimal digit representation has been so processed.